From Lab Rack to Trading Floor

A Journey in Local Intelligence

WHITEPAPER | Local LLMs, edge intelligence, and controlled AI infrastructure

Author
Tom van de Ven

Publication date
August 2026

A practical whitepaper on local intelligence, privacy-aware AI infrastructure, and the economics of running specialized models close to the work.

Technology & Innovation | Local AI/LLM experimentation

1. Table of Contents

2. *Building a thing* 3
3. *Lab Rack Hardware* 4
4. *Local LLMs* 6
5. *Creating an LLM powered trading agent* 7
6. *Token Economics* 9
7. *The Trade-off: Capability vs. Specifics* 11
8. *Future Outlook: Distributed Intelligence* 12
9. *The Present Reality* 14
10. *A Philosophical Shift* 15
11. *Local or global LLMs: Who can you trust?* 16
12. *Evolving the trade bot...and testing it!* 17
13. *Let's talk business cases* 19
14. *Limits and Uncertainties* 20
15. *What changed for me?* 21
16. *Epilogue: The Lab Rack Today* 22
17. *Appendices* 23

Appendix A. Complete Features & Requirements Overview **24**

Appendix B. Prompt History **26**

Appendix C. Testing Guide **32**

Appendix D. Complete Testing Overview **35**

Appendix E. Test Results **37**

2. Building a thing

It started, as many projects do, with a project someone else was busy with. In this case I came across Lab Rax: a project for 3D printing your own rack that can house smaller server parts. Someone scaled it down to a 5 inch rack. It looked fun and it could house some Raspberry Pi development boards. Beign a fan of 3D printing, it was the 3D printing that caught my interest. I printed every part, looked for the right slots to house 2 Raspberry Pi 3's and one Raspberry Pi 4 that I had laying around. Added a slot for a 4 port router and everything was good to go.

Building something before having a good use for it is something a lot of engineers may experience. It is not necessery a bad thing but you might end up with a room full of ... well ... things. In this case the building proces gave me time to think about how to put this small compute power box to good use. There was one thought that wouldn't leave me alone. I was tired of my smart home depending on hyperscaler (read: Cloud) infrastructure, and the endless cycle of data collection and privacy concerns.

- How many times per day is my smart speaker was listening, processing, uploading?
- How many queries about my habits were being logged somewhere?
- How many algorithms were learning about me without my understanding?

The plan was simple on paper: run Home Assistant on the Pi4, automate my house locally. But I wanted more than just if-this-then-that automations. I wanted the hyperscaler's intelligence. I wanted my house to understand context. To make this meant running a language model locally. No cloud calls. No tokens being counted against my account. Just pure, local computation.
Next to that I wanted to find out how "smart" my local intelligence can actually be. Adding to smart home automation I wanted to see if a simple stock trader could be created that uses the LLM to parse the news and makes trading decisions based on that news input. It must maintain a portfolio of stocks and think about buying, selling and creating the buying space as well.

Looking up what LLMs can run on my type of hardware, I quickly found out that TinyLlama was a good choice. With only 1GB of RAM it needs to be small enough to be responsive. It does not have to give instant responses with the speed and might of hyperscaler LLMs. Small audio commands only needed to be converted into simpel actions: switch a light on and off, set a temperature, play a song. The play a specific song being the most complex one.

The idea behind setting this up is to find out what hardware is needed to do the minimum amount of LLM work. This situation did not require an LLM that can code banking applications, creating recipes based on a picture of my fridge and defining a four-week holiday roundtrip with routing, overnight stays and highlights to visit all at the same time. In my case I only need parsing of simple commands and finding out if my home automation system can work with. At best some optimisation of my home automation may be done.

The open-source community has been working on this with distilled models, models specifically designed to fit on constrained devices. The barrier to entry was dropping rapidly.

3. Lab Rack Hardware

Building a thing involves hardware. I already described the lab rack that I created. Let's focus on the rest of the hardware that is in the lab rack. This is the stuff where everything is going to run on. In the lab rack I have room for three raspberry pi's and one switch to connect the pi's to a local network and to have them communicate with each other.

Figure 1. My lab rax case filled with hardware

In my case (See Figure 1) there are two raspberry pi 3's and one raspberry pi 4. The switch is a 5-port Gigabit Switch. It uses one connection to link the lab rax to the outside world (a local network and/or the internet). The other connections are used to connect each raspberry pi and optionally a laptop to connect locally through terminal windows with the raspberry pi's.
The power supply to the raspberry pi's is through an USB hub and the switch is using its own power supply. There is an option to power all raspberry pi's with Power-over-Ethernet (PoE) through the switch. That would require adding shields to the pi's but leaves you with one power supply for convenience.
This specific setup is what I had "lying around" and not specifically chosen for its performance. In this document I will discuss other options that might perform (a lot) better. On the other hand with this set of relatively low end processing power and memory I can demonstrate what is already possible.

4. Local LLMs

Installing a local LLM on that Pi3 was fairly straight forward. A fun fact is that I used an LLM (in this case Claude CLI) to find out all I needed and point me to the locations for downloading parts as well as how to integrate and install where needed. Using an LLM to setup an LLM!

An then ... it just worked—when I asked it the first question and it responded without any network, without any cloud service being involved, character by character, on my screen. It was thinking. Right there, in my house, on hardware I owned, using electricity I was paying for.
For the first time, I wasn't shipping my home's state to some cloud provider. I was asking my own hardware: "Based on the current temperature, time of day, energy prices, should I run the dishwasher now?" And it would thinken on this (yes it took some time and the words appeared litteraly word ... for ... word.

No token counting.
No monthly surprise bills.
No data leaving my network.

The implications started to sink in immediately. Every automation decision was now private. Every context was preserved locally. I could ask it to learn my preferences, my habits, my needs, without any of that information being sent to a third party.

We've been trained to think we need the most advanced AI systems. Claude, GPT-4, the latest and greatest. But the truth is, most of the time, we don't. My home doesn't need world-class reasoning. It needs specific reasoning. It needs to understand my home, my preferences, my energy costs. In general it is possible that a 7B parameter model with focus on only the aspects you need can outperform a 70B general model.

Running TinyLlama with the llama.cpp interface (a 1.1B parameter model) on the Pi3 taught me this lesson even more directly. I'd bought that Pi3 to play around with in the past. It was so old, so underpowered. But TinyLlama was specifically designed to run on exactly this kind of hardware. And it worked. In my netwerk I now had: **Llmpi**!

I started experimenting with it. I'd ask it general questions, and my Llmpi gives answers. They are sometimes rough around the edges, occasionally inaccurate, but for many practical purposes, they are good enough. Better than good enough—they are useful. The model understands context in my queries. It can follow multi-step instructions. It can write code, albeit simpler then Claude can produce.

I was under the impression that LLMs are almost exponentially scaling with what it can do. It is therefore also exponentially scaling with hardware that it needs, storage that it needs and thus the costs that are involved. With this project TinyLlama breaks that model for me. It shows me that capability is multidimensional. A smaller model with better fine-tuning for a specific task can be more useful than a massive general-purpose model.

5. Creating an LLM powered trading agent

Now that the basics are in place it is time to take the step to put my small local LLM to good use. I want to have one overview of how my house is doing, what kind of energy I am producing/using (and in the future learn from this) as well as see the intelligence of my LLM continuously put to work.
The other idea I had is to make a stock trading agent with my local LLM and trade on a paper account (for obvious reasons). In the end I want to see if my local LLM powered stock trading agent can do better then a real human (me!). In Appendix A. Complete Features & Requirements Overview you can find the details of what I created.

With the stock trading agent I do not expect to get rich, I'm realistic about that. Plenty of algorithms have failed at day trading. But because it is a perfect testcase for what local LLMs could really do, I am building it:

- A controlled environment.
- Measurable outcomes.
- Clear success and failure metrics.

I set up a paper trading account on Alpaca and started building: News fetcher. Technical analysis. Risk management. All the pieces. But here's what surprised me:
the trading bot does not need to be brilliant. It needs to be consistent.

I feed it news articles about specific stocks—only the most relevanty. And minimal technical indicators—not the full arsenal of indicators, just what mattered. RSI, trend, volume. That's it. No MACD divergences or Bollinger Band squeezes or Fibonacci levels. Just the essentials. And I let TinyLlama analyze them with instructions tuned specifically for daily trading reasoning, not essay writing. Just: "Here's news and technicals, make a binary decision (well almost: BUY|SELL|HOLD) with a confidence rating."

For the development process I used an the Claude CLI to help me out. I'd write a version, test it against my paper trading account, see where it failed, and adjust. After a couple of iterations I found out that the application built by Claude was too sophisticated for TinyLlama. TinyLlama would get confused, its responses would ramble, the confidence ratings wouldn't match the actual quality of the signal. I learned to be more concise, more specific, more forceful about what I wanted.

I had to solve the problem of parsing TinyLlama's responses. Sometimes natural language, sometimes a mix. I created a robust parser that could handle multiple formats and extract the decision even when the model didn't strictly follow my instructions. That taught me something important:
working with smaller models means accepting their quirks and handle them.

The bot came to life on my Pi3 and it started analyzing stocks, making trades, managing positions "*intelligently*". I watched it buy on high-confidence signals. I watched it cut losses when the LLM said the conviction was there. I watched it hold when it said to
wait. I watched it rebalance when a better opportunity appeared.

And here's the thing: it worked. Not because TinyLlama is some incredible LLM but because the task was specific enough that a small model could handle it perfectly. The model didn't need to understand poetry or physics or philosophy. It needed to understand financial news and technical patterns. It was like training a specialist versus hiring a generalist.

It is running in the Lab Rack now and doing its trading. It is actually fun to watch making its decisions and doing its trading. The P&L isn't spectacular but seems to increase the portfolio value. The bot is making intelligent decisions. It wasn't just random luck. The trades it makes have reasoning behind them. The confidence ratings correlate with actual outcomes. The risk management worked as intended.

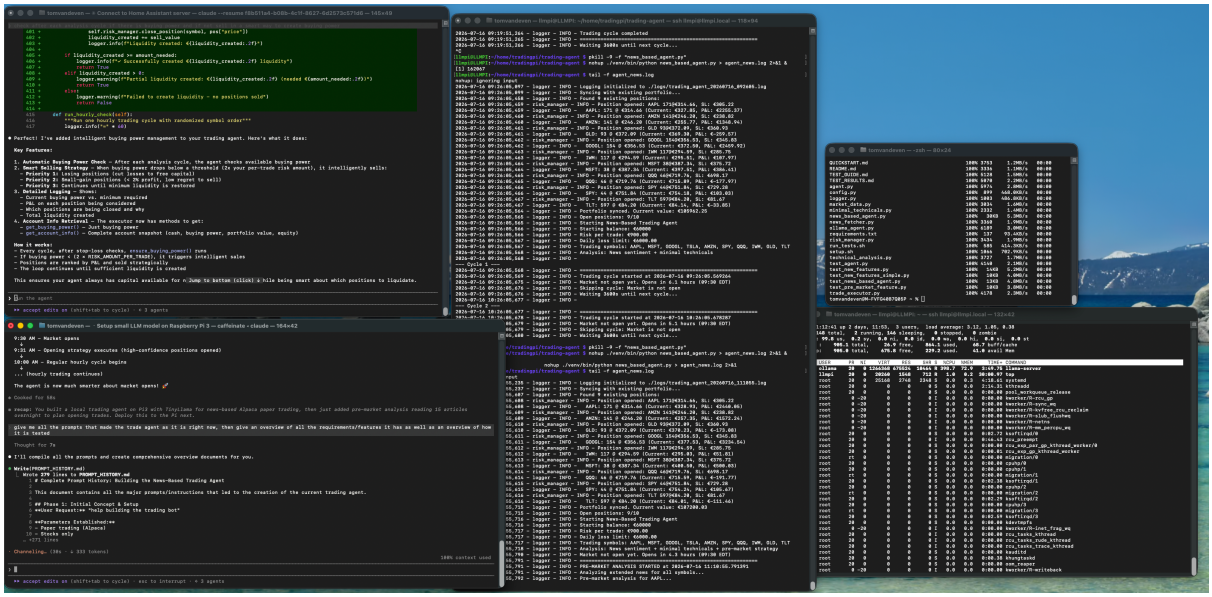

Figure 2. How I keep an overview of all the developments going on on the different pi's with each their different purpose (as well during development as during operation).

Building the stock trading agent was done in a couple of days. Using multiple terminals with their own Claude doing their own things made the development of this also a nice agentic experience. It can interact with my different Raspberry pi's through the command line interfaces where needed. I am creating the agent locally on my desktop and testing it there before deploying it on the Raspberry pi's. In the appendices there is a full description of the complete feature list, the prompts I used as well as a test methodology, implementation and test results.

6. Token Economics

At this point I started thinking about tokens in a way I hadn't before.

Every API call to Claude, every request to GPT-4, every prompt I enter is counted and billed. And here's what bothered me: How do tokens map to effort the we know (think power, effort)? We have some esitmates but no exact numbers that we can calculate with.

"*About 4 tokens per word.*"

This is what we have to work with. Creating a prompt is dependant on so many variables (in wording, in total words, in language, in the person creating it, its intent and background, etc. etc.) When you're running millions of inferences across thousands of users, those estimates compound into uncertainty. How are they calculated for a sentence? How about special characters? Spaces? Are they rounding? Batching? Applying compression? What is the LLM using internally for reasoning? The pricing models are opaque, and they're getting more opaque as models get larger and as usage patterns become more complex.

I started tracking my token usage. I set up alerts. I analyzed my spending patterns. And what I found was interesting to say the least: I was paying for the same reasoning multiple times. I'd ask Claude a question about architecting then ask it a follow-up question about implementation. That was potentially two separate token counts instead of a single conversation. And when I multiply that across hundreds of developers in a team, across thousands of queries per day, the waste becomes astronomical. When the outcomes of your conversations are logged in a document, the LLM can reuse that the next time you ask things, a lot of tokens could be saved in this way. But that is not how LLM’s are working.

The costs are staggering. A developer running inference-heavy applications can spend thousands per month. Developers can be charged $15,000 in a single month because they had a bug that caused their application to generate repeated queries. A company scaling trading bots, content generation, customer support automation? Tens of thousands. Hundreds of thousands. I can imagine situations that accumulate to millions per month for LLM inference costs.

It's a beautiful business model for large hyperscalers offering their LLMs like OpenAI and Anthropic. Recurring payements and pricing that can be adjusted without the customer really understanding what they're paying for. But it creates a ceiling on what local businesses can build. A startup building AI applications can't scale without
worrying about token costs becoming their largest expense.

Running a local LLM—even a small one like TinyLlama on a Pi3—solves tokens. No meters. No bills. Just electricity and hardware. The variable cost per inference rounds to essentially zero. You could run a million inferences in a month and spend maybe $10 in electricity.
Let us compare this with running for example Microsoft CoPilot: The stock trading agent runs 30 x 24 x 60 x 60 seconds/month = ~2.6M seconds. Let’s say for a large frontier model this is 15 token/second, totalling ~39M tokens per month.

Estimated Cost Breakdown for 39M Tokens:
The baseline rate for standard frontier models (like Claude 3.5 Sonnet or GPT-4o) averages roughly $3.00 per million input tokens and $15.00 per million output tokens.Assuming a typical agentic/chat

workflow split (85% input context/repository indexing and 15% output generated code), here is how the numbers stack up: 15% output generated code), here is how the numbers stack up:

Model Tier Example	Input Cost (Per 1M)	Output Cost (Per 1M)	Total Estimated Cost for 39M Tokens
Lightweight Models *(e.g., Haiku / Flash)*	~$0.30	~$1.20	**~$17 to $25**
Standard Frontier *(e.g., Sonnet / GPT-4o)*	~$3.00	~$15.00	**~$180 to $200**
Reasoning / Heavy Models *(e.g., o1 / Opus)*	~$15.00	~$75.00	**~$900 to $1,000+**

For lightweight models this is comparable to running a local LLM. Even with an enterprise licence this amount of tokens quickly becomes a costly affair.

Why is not everyone running local LLM's you might wonder! Here's the catch: you have to know what you're doing with local LLM's:

- You have to manage the infrastructure.
- You have to handle the complexity.

And that's where most organizations do not want to go or can go.

I also started thinking about the accounting problem differently. Not just for business reasons, but because it's genuinely difficult to measure precisely. How do you count tokens for vision inputs? For audio? For video? Different models count things differently. OpenAI's tokenizer isn't the same as Anthropic's, which isn't the same as Meta's. When you're into a heterogeneous ecosystem of models, it becomes nearly impossible to predict what you'll actually pay.

And this uncertainty is only going to grow. As models get bigger, as we layer them (using one model's output as input to another), as we work with multimodal inputs, the token accounting becomes more fuzzy. I predict that we will have entire teams dedicated just to understanding and optimizing their LLM token usage. It's already happening. There are many initiatives out there that compress and optimize your prompts before executing them (think Caveman or a simple (older LLM) as pre-stage before going to the "expensive" LLM like LLMLingua). It clearly shows the benefits of lesser token usage but how much of your prompt information is now "lost"? We can also be trained to do better prompting. Become really skilled prompters is almost the same as becoming a very code developer. Where is then the gain in prompting vs. coding?

It is both an opportunity and a warning. An opportunity because there's clearly demand for solutions that make this transparent and optimizable. A warning because the complexity suggests that cloud LLM pricing has become a hidden costs in the AI economy.

7. The Trade-off: Capability vs. Specifics

You can't run GPT-4 on a Raspberry Pi. You can't run Claude on it as well They are just too large, too complex. Combine that with not one but many users and they require data centers, thousands of GPUs, specialized infrastructure that costs millions to build and maintain.

But do you need them?
Most of the time, no.

Here's what I learned building the stock trading bot: general-purpose intelligence is expensive. It's overkill for most tasks. What you actually need is specific intelligence.

My stock trading bot doesn't need to write poetry or answer trivia or indulge in complex reasoning about geopolitical events. It needs to:

- Parse financial news efficiently
- Extract relevant signals
- Calculate technical indicators correctly
- Make yes/no/hold decisions
- Assign confidence ratings
- Manage positions with risk awareness

All of that? TinyLlama handles it beautifully on a Pi3. Not just aduse it's been given instructions that are hyper-specific to that task.

That's the future I see: not everyone using Claude or GPT-4, but everyone running something small. Something that is tuned to their specific needs.

I started experimenting with fine-tuning a smaller model specifically on financial news. I collected historical news articles and paired them with market outcomes. I trained a small adapter on top of TinyLlama. The results started making better trading decisions not because it was large or more capable in a general sense, but because it had been exposed to the specific patterns in financial news that correlate with price movements.

The lesson was profound: a 3B parameter model fine-tuned on your specific domain can outperform a 70B general model by orders of magnitude. Not in all tasks obviously, but in your tasks: the things you actually need to do.

This has huge implications. It means that the future of AI isn't dominated by whoever builds the biggest model. It's fragmented and specialized and full of smaller models tailored to specific domains.

8. Future Outlook: Distributed Intelligence

I've started imagining what this could look like scaled up, and the possibilities excite me more than any new frontier of general AI capability.

Picture a development team: Fifteen people. Instead of each developer sending every code question to Claude on the CLI, consuming tokens, paying per inference,
they have a local code-intelligence box. A stack of Mac mini's with an M4 processor in their office. It runs a 7B parameter code model, fine-tuned on their codebase and their company's coding conventions. When someone asks, "How do we handle authentication in this project?" it answers instantly. The best thing is that it runs: locally, no token cost with no network latency or privacy concerns.
Even with the cost of electricity a group of developers can match or spend less money on tokens in this way. Maybe even better: it is local. In some cases this is an even bigger trigger then the money saved.

That's not science fiction. It's already possible. I could build this hardware setup and run the models. For my one-man development team (myself) it is already running and working great.

Scale it further: a company with multiple teams could run a rack of serving dozens of teams. Options like the Nvidia DGX Spark are great as an intermediate solution. Or they could go full enterprise with external GPU boxes (A100s, H100s) configured as a local inference cluster. Developers query it like they'd query any internal service. The economics work out because you're not paying per token. You're paying once for hardware and you can keep running thousands of inferences.

Here's where it gets really interesting: the Mac minis with M-serie are uncannily good for running LLMs. Apple did not design them without token metering in mind, but they're phenomenal for it. The unified memory architecture, the efficiency of the ARM design, the integration of the neural engine—it's almost like Apple inadvertently created the perfect "LLMmini". An M4 costs $600-1000 and can reliably serve a 13B model with good latency. The economics of that are hard to beat.

Or take it in a different direction: Arduino boards with AI chips. Qualcomm's Snapdragon X Plus. Edge devices that can run specialized models locally:

- A security camera that identifies threats without uploading footage.
- A factory anomalies without calling home.
- A medical device that assists diagnosis without sending patient data to the cloud.
- A manufacturing robot that adapts to new tasks without querying a cloud service.

Or—and here's what I find most compelling—you could have a Mac mini on your desk right now next to your workstation. With the M4 processor, sixteen cores, enough memory to run a solid 13B model comfortably this is set up once to run everything locally: Code generating, documentation, etc. It is done all local and all free after the initial hardware investment.

I started modeling the economics. A company spends $2,000 on a Mac mini. It runs inference for two years before it needs upgrading. That's $1,000 per year. Let's not forget we need extra infrastructure expertise, electricity. The infra expert needs to be payed and just as in the past, depending on the size a server park needs to be maintained. There are costs involved here as well. Compare that to a company that's using cloud LLM APIs and spending token costs. The savings are not marginal. They're transformative.

But the transition requires expertise. You need to:

- Select the right model for your task
- Manage hardware and infrastructure
- Handle model updates and versioning
- Implement proper caching and optimization
- Monitor performance and reliability
- Potentially fine-tune models for your specific use case

Most organizations can't do this alone. The infrastructure problem is an interesting business opportunity: the infrastructure-as-a-service-play for local LLMs. Someone manages the hardware, the models, the updates and you just query an API. It's still local to your organization (or at least not shared with the cloud), but you don't have to run it yourself.

There's a company in that idea (maybe several).

I've also been imagining smaller deployments:

- A team of data scientists using a server for their ML experiments.
- A law firm wanting to analyze contracts without sending them to OpenAI's servers.
- A healthcare provider wanting to flag potential diagnoses without worrying about HIPAA violations.
- A manufacturing company wanting real-time anomaly detection on the falatency.

The use cases are endless. And the common thread is the same: you decentralize the AI. You need something specific, something local, something you control.

Here's what I see when I think about the next five years:

Organizations have multiple models running locally. A 3B model for coding. A 13B model for writing. An ensemble of models that can tackle different problems efficiently.
The cloud still has a role, but only for specialized needs. GPT-4 handles problems that truly require it, and Claude supports deep reasoning that smaller models cannot. Most inference, however, runs locally.

Edge devices are running specialized models. Your camera has AI. Your fridge has AI. Your car has AI. Most of it is running locally because that's cheaper and more private and lower latency.
Organizations have hired "LLM infrastructure engineers" specifically to manage their local models. They fine-tune models. They select models. They optimize inference. They manage model versions.

The token metering problem has gotten so out of hand that companies can’t work with it. They're going to either running localyl or they're budgeting huge but fixed amounts of money per year for cloud inference.
The conversation about AI shifts from "how do we build bigger models" to "how do we deploy models efficiently." Efficiency becomes the key differentiator. Companies compete on inference speed, model size, power efficiency and local LLM infrastructure.

This is the future I'm betting on.

9. The Present Reality

Right now, in my lab rack, the balance is struck imperfectly. I'm reasoning via the CLI. I'm using TinyLlama for specific tasks like trading. I'm using larger models in the cloud when I need them, local models when I don't.

The token costs are real. But I've also learned that some of that spending was unnecessary. I was reaching for a sledgehammer with the latest and greatest LLM when I only needed to analyze one paragraph of text (maybe GPT-3 would have been fine). I was asking Claude to do tasks that a specialized model would have done better.

The trading bot running on the Pi3 costs me nothing in tokens. It costs me the electricity to run it. And it works: it trades intelligently, it manages risk, it learns. All for the price of some electricity.

My home automation setup costs me nothing in inference: again only some electricity. My Home Assistant is running on the Pi4, making intelligent decisions about my house, all without any cloud service knowing my habits or my preferences.

I'm at a point where I can see the all these LLM opportunities clearly. The local LLM path is viable and economical. The cloud LLM path is convenient but increasingly
expensive. The optimal path forward probably uses both.

10. A Philosophical Shift

What strikes me most about this journey is the philosophical shift. The starting point is the belief that bigger is better: better models, more parameters, more capable systems. In some cases, that's true. General reasoning, creative writing, complex problem-solving, novel domains—those still benefit from the largest models.

But for most of what we actually do a 7B code model that's fine-tuned on your codebase beats a 70B general model that knows everything about everything. A 3B trading model that knows financial news beats a 13B model that also knows poetry and philosophy and ancient history.

The future isn't one winner. It's not "everyone uses the latest GPT version for ever. Is is also not that we are going for a fully specialized, local, distributed-across-hardware-sitting-next-to-our-laptops-situation (in all our offices, on all our factory floors).

Some tasks will still require calling out to the cloud LLMs. This can be truly complex reasoning or really novel problems. These are cases where token cost doesn't matter because the value is so high. Financial analysis for mergers. Drug discovery. Strategic planning. Fine, pay the tokens. The value is worth it.

But everything else? The everyday stuff? The routine decisions? Test case generation? Content creation? That should be local.

I had a moment recently where this philosophical shift came to life on a problem that required creative thinking. I opened Claude on the CLI and started asking questions. The back-and-forth was valuable. The model's perspective helped me think about the problem differently. I probably spent 50,000 tokens in that session. It cost me about $3.

But then I thought: if I had this conversation every day, that's $9,000 a year. For a single person's creative work. That's the beginning of being prohibitive.

If instead I had a local 13B model running on a Mac mini, I could have the exact same conversation for free. Sure, the responses might be slightly less polished, but I could be having them constantly without worrying about cost.

That's the shift. That's the future I see.

11. Local or global LLMs: Who can you trust?

The future I see is also deeply uncertain. Will companies embrace LLMs so that it become so cheap and so good that it doesn't matter? Will regulations around data privacy force local deployment, or will they remain optional? How will token pricing evolve? Will companies like OpenAI and Anthropic maintain their pricing power, or will competition drive it down? Will there be something so much better that token cost becomes irrelevant?

I don't know. But I do know this: the technology to do this locally is getting better and the hardware is getting cheaper

I'm betting on a future where local and cloud coexist. Where it's normal to have multiple models running on your personal infrastructure, your team's infrastructure and your company's infrastructure. A future where token costs are actively managed instead of passively paying them. The question isn't "should we use an LLM?" but "where should we run this model?"

There's also the possibility that this whole thing is premature. That cloud inference becomes so cheap and so fast that the economics don't favor local deployment. That GPT-5, 6, 7 and higher or Claude Next or whatever comes next is so good that the cost is irrelevant compared to the value they get. That's possible.

But there's also the possibility that I'm right. That local inference becomes the default for most use cases. That companies realize they can save millions by running local models.

The truth is probably in between. Some tasks fully local and some tasks always in the cloud. Ther is going to be a huge gray area in the middle where economics, privacy, latency and capability all trade off against each other.

What I do know is that I built something. I took control of a piece of my infrastructure. I'm running an LLM in my house, in my rack, on my hardware. And it's working. It's autonomous.

That feels important. That feels like the future.

12. Evolving the trade bot…and testing it!

The system didn't stop evolving after those days of running it. I keep on learning.

I realized that the trading bot was missing something important: ths own trades. After each position closed, it had valuable data about whether its decision was right or wrong. I added logging that captured every trade, every decision, every confidence rating, and the actual outcome.

Then I started analyzing the logs. Which news patterns led to good trades? Which technical indicators actually mattered? Where was the model overconfident? Where was it too conservative?

The bot got better. Not dramatically better. This isn't a success story about 100% returns or beating the market. It's a story about incremental improvement. The average winning trade went up marginally but it became more sophisticated.

And here's the key thing: I did all of this without spending a single token. Every improvement came from local experimentation, local analysis, local learning.

I also started to think about the system's limitations. TinyLlama sometimes misunderstood complex news articles. It would misidentify what the article was saying about a stock. When I fed it conflicting signals, it would sometime mess up the confidence ratings.

The system evolved from a simple first version into something more robust, more sophisticated. And it was all happening locally, without external dependencies, without token costs.

Not only did I add new features and improvements, I also added test cases for the existing setup. In order to create new versions I wanted to make sure there were no disturbances in working solutions. Setting up testing really helped getting confidence and trust in the solution (check Appendix D. Complete Testing Overview for more details). Planning out testing, letting the LLM look at the test results (see Appendix E. Test Results) and giving me feedback, kept the human in the loop but not with a huge effort of having a test engineer on site. See

Working with an LLM grows on me. Over time I started to trust the LLM more and more in helping me create things. This is where we are lulled to sleep. This is the moment where I really started to focus on a solid test methodology. A big danger lies around the corner: ask your LLM to do the testing for you. If you trust your LLM why not let it test? This is exactly where things can go wrong. If the same entity that builds also tests, there is a high risk that the tests will passnot finding defects. At best it builds a test framework that can help filtering out regression with future changes. Really testing if your requirements have been implemented in a correct way is something different.

The fun thing is that an LLM can help you with testing. You have to do the creative and quality work (see Appendix C. Testing Guide for some details on testing the News-based trading agent). Just as with spec-driven development, you have to tell the LLM what your test specs are. Why do you want to have stuff tested? This is where your test methodology must guide you. Asking an LLM to create test cases for specific scenario's is perfect. Reviewing what it creates is imperative (just as with regular code).

Do not think that if an LLM can build something that works, it is what you want and behaves how you want over time and in all the crazy edge cases it can end up in. Software development is a skill and test engineering is definitely part of that skill. Trusting the LLM to do all might be a little naïve.

13. Let’s talk business cases

As I've been thinking about scaling this, the business case has become clearer and more compelling.

Let's say you're a financial services company. You have 100 traders or analysts. Each one is using cloud LLMs, paying an average of $200/month in tokens. That's
$240,000 per year, just in LLM costs. This is a conservative estimation and only for a simple business case calculation.

Now imagine instead that you run a local inference server. This can be a rack with managed infrastructure. Capital cost: maybe $50,000-100,000. Annual operating cost: maybe $5,000-10,000 in electricity and maintenance.

Suddenly your token bill is gone. Your LLM costs dropped by 90%. Take into account that the amount of tokens used in the future is also rising (maybe even exponentially) we can quickly calculate that over a five-year period, you save $500,000 to $1,000,000 or more. This is costs alone and not taking privacy and unlimited LLM usage into acount.

That's not a nice-to-have. That's business-transformative.

But here's the problem: most companies can't do this themselves. They don't have the expertise. They don't have the infrastructure team. They don't want to hire someone just to manage their inference servers.

Which is where the business opportunity lies. Someone could offer:
“We'll provide the hardware, the management, the model updates, the security. You just use it. You get all the benefits of local inference without any of the pain of managing it.”

That's a company. That's multiple companies, probably. There's the managed service play, the hardware play, the fine-tuning play, the model selection play.

14. Limits and Uncertainties

But I want to be honest about the limits and uncertainties.

Local LLMs won't replace cloud LLMs. There are tasks where you need the big frontier models out there and privacy/security is secondary to capability:

- Strategic decision-making (although this one can be debatable in terms of privacy/security)
- Complex reasoning
- Novel problems

Also, the local LLM ecosystem is fragmented and immature. There's no clear standard. Different models work best for different tasks. Selecting the right one requires expertise. Combining models requires engineering and managing the infrastructure around the models.

And there's a real possibility that cloud LLMs get so good and so creative. If OpenAI figures out how to run inference at 10x lower cost, or if their next model is 10x better such that the token cost is irrelevant, the whole calculus changes.

I'm also uncertain about regulatory futures. Right now, there's no regulation around where LLMs have to run. But what if regulations require data processing to stay local? What if GDPR or HIPAA get interpreted to require local huge tailwind for the local LLM market. Or what if regulations gothe other way, requiring use of certified cloud providers? That would be a headwind.

The token accounting problem might get solved. Someone at one of the hyperscalers might publish a detailed whitepaper explaining exactly how tokens are counted. The industry might converge on a standard. But I think that's unlikely.

15. What changed for me?

What's changed for me is more subtle than just economics or technology. I'm more aware of when I'm outsourcing my thinking to AI. I'm more aware when I ask Claude certain questions and I try beforehand to think of the direction answers must converge towards. I use local models for routine work so I can save my token budget for the hard problems.

I'm also more aware of privacy. It matters to me that my home automation system isn't calling Amazon's servers. It matters that my trading bot isn't sending my trading ideas to external services. It matters that my code isn't instantly shared with a hyperscaler for whatever reason (even though I trust a company like Anthropic with my code, it's still my code, and I prefer it stays private). And I'm more aware of the future I want to build. I want to build systems where the intelligence is local, the data is private, and the economics are transparent.

16. Epilogue: The Lab Rack Today

The lab rack still sits beside my desk, though it's grown. I added additional storage. It's become this modest hub of local intelligence.
The Pi3 still runs TinyLlama, still trades. The P&L has been positive, though modest. The point was never to get rich. The point was to prove that you could do sophisticated things with modest hardware and the right approach. I am also still paper trading, trust does not go that far that I trust it with my own earned money.

The Pi4 still manages my home. I've added more sensors, more automation suggestions from my LLM are implemented in my household without a cloud solution toughing my home.
The future of AI might not be in the cloud at all. It might be sitting on my desk, or in my rack, or on a company server, doing the specific work that needs to be done, at the cost of hardware and electricity, not tokens.

That future is worth building toward.
And I'm just getting started.

17. Appendices

Technical reference material for the news-based trading agent: requirements, prompt history, testing guidance, testing overview, and test results.

Appendix	Purpose
A	**Complete features and requirements overview.**
B	**Prompt history and implementation evolution.**
C	**Testing guide and practical validation commands.**
D	**Complete testing overview and coverage summary.**
E	**Test results and feature validation status.**

Appendix A. Complete Features & Requirements Overview

Summary

A sophisticated stock trading agent that combines news sentiment analysis, technical indicators, local LLM reasoning, risk management, and smart portfolio rebalancing.

Area	Clean Markdown-styled summary
Architecture	Raspberry Pi cluster: Pi3 for LLM inference, Pi4 for orchestration.
Market	Alpaca paper trading, stocks only.
Language model	TinyLlama via Ollama.
Budget	€60,000 with 1.5% risk per trade.

Core Requirements

Requirement area	Cleaned requirements
Market access & data	Alpaca paper trading, real-time quotes, position tracking, hourly bars via yfinance, account information, and Yahoo Finance news.
Risk management	€60,000 starting balance, €900 risk per trade, 10 maximum positions, €6,000 daily loss limit, 15% maximum position size, and automatic 3% stop loss.
Technical analysis	RSI, SMA 9/21, Bollinger Bands, volume analysis, trend detection, and minimum bar requirements.
LLM decision-making	TinyLlama through Ollama, JSON response parsing, fallback text parsing, and confidence scoring.
Trading logic	BUY at confidence ≥ 0.65, SELL at confidence ≥ 0.70, HOLD below threshold, and dynamic rebalancing when position limits are reached.
Portfolio management	Startup portfolio sync, position P&L tracking, stop-loss levels, cash, buying power, equity, and position count.
Pre-market strategy	15-article overnight news analysis, market bias detection, confidence averaging, and 9:30 AM ET opening execution.
Bias elimination	Randomized symbol order every cycle to avoid first-stock and sequence bias.
Error handling	Graceful fallback for market data, news, LLM timeouts, parsing failures, and restart recovery.

Feature-by-Feature Breakdown

Feature	Status	Implementation	Impact
Alpaca integration	Complete	REST API and paper trading	Core trading capability
News fetching	Complete	3 articles during trading, 15 pre-market	Sentiment analysis
Technical analysis	Complete	RSI, SMA, Bollinger Bands, volume, trend	Signal generation
Risk management	Complete	Position sizing, stops, limits	Capital preservation
Portfolio sync	Complete	Alpaca position loading	Restart resilience
Dynamic rebalancing	Complete	Worst-position closing	Portfolio optimization

Market hours check	Complete	9:30 AM–4:00 PM ET	Trading compliance
Logging and audit trail	Complete	Decision, reason, confidence, and portfolio logs	Debugging and traceability

Configuration Parameters

Category	Parameter	Value	Purpose
Capital & risk	STARTING_BALANCE	60000	Initial paper-trading budget.
Capital & risk	RISK_PER_TRADE	0.015	1.5% of account value per trade.
Capital & risk	MAX_POSITIONS	10	Maximum number of concurrent positions.
Trading	CHECK_INTERVAL	3600 seconds	Hourly trading cycle.
Technical	RSI_PERIOD / SMA_SHORT / SMA_LONG	14 / 9 / 21	Core indicator configuration.
News	REGULAR_NEWS_LIMIT / PRE_MARKET_NEWS_LIMIT	3 / 15	Article volume during market and pre-market analysis.
LLM	OLLAMA_MODEL	tinyllama	Local language model used for reasoning.
Thresholds	BUY / SELL confidence	0.65 / 0.70	Decision gates for trade execution.

Workflow & Cycle

Stage	Time window	Activity
Pre-market analysis	4:00 PM–9:30 AM ET	Fetch extended news, analyze all symbols, calculate market bias, and store opening strategy.
Market open	9:30 AM ET	Execute high-confidence opening positions once, then transition into the regular cycle.
Hourly trading	10:00 AM–4:00 PM ET	Analyze randomized symbols, fetch news and technicals, ask the LLM for decisions, enforce risk limits, and log activity.
Market close	4:00 PM ET	Stop trading and return to pre-market preparation.

Success Metrics

Metric	Target	Current status
Risk compliance	100%	Enforced through position sizing, loss limits, and stop losses.
Daily loss limit	Never exceeded	Validated in test scenarios.
Market-hours compliance	100%	Trading blocked outside 9:30 AM–4:00 PM ET.
Portfolio survival	Clean restarts	Existing positions are loaded on startup.
LLM reliability	<5% parse errors	Fallback parser handles non-JSON responses.
Confidence calibration	Signals align with outcomes	Tracked through logs and reviewed over time.

Deployment & Runtime

Area	Details
Hardware	Pi3 with 1GB RAM for TinyLlama inference; Pi4 with 4GB RAM for orchestration; custom 3D-printed lab rack.
Dependencies	Python 3.13+, alpaca-trade-api, yfinance, pandas-ta, pytz, and local Ollama runtime.
Runtime command	Run the agent from the trading-agent virtual environment using nohup and log output to agent_news.log.
Monitoring	Tail the runtime log, filter trade events, and query Alpaca paper positions when needed.

Testing Coverage

Test area	Coverage
Core unit tests	Technical indicators, decision parsing, risk management, news fetching, LLM handling, and integration workflow.
New feature tests	Market hours, risk manager functionality, confidence thresholds, account info, randomization, and LLM confidence bounds.
Pre-market tests	Extended news fetching, strategy structure, strategy execution, market bias calculation, and duplicate prevention.
Operational validation	Portfolio sync, stop-loss behavior, market-hours enforcement, and logging/audit trail.

Future Enhancements

Enhancement	Status
Multiple asset classes	Not implemented.
Sector-based trading bias	Not implemented.
Volatility-based position sizing	Not implemented.
Advanced order types	Not implemented.
Machine-learning confidence calibration	Not implemented.
Real trading	Not implemented; current setup remains paper trading only.
Performance attribution analysis	Not implemented.
Model fine-tuning and multi-source sentiment	Not implemented.

Appendix B. Prompt History

This appendix captures the major prompts and instructions that shaped the news-based trading agent, rewritten as a clean implementation history.

Summary of Evolution

Phase	Focus	Key addition
1	Foundation	Project structure, configuration, and market data.
2	LLM integration	Technical analysis and TinyLlama decision prompts.
3	Risk management	Position sizing, stop losses, and trading limits.
4	Intelligent trading	Confidence thresholds and profit-taking logic.
5	Market hours	Trading limited to 9:30 AM–4:00 PM ET.

6	State management	Portfolio synchronization on restart.
7	Portfolio optimization	Dynamic rebalancing and position cycling.
8	Scaling	Budget increased from €10,000 to €60,000.
9	Liquidity checks	Buying-power validation before trades.
10	Bias elimination	Randomized symbol analysis order.
11	Pre-market strategy	Extended overnight news analysis and opening strategy.
12	Production readiness	Final tuning, confidence logic, and operational hardening.

Key Prompts by Impact

Category	Prompt themes	Impact
Highest impact	Confidence-based trading; pre-market analysis; dynamic rebalancing; market-hours validation.	Improved signal quality, gave the agent an overnight information advantage, kept the portfolio active, and prevented invalid trades.
Technical foundations	Risk management; portfolio sync; buying-power checks; randomized analysis.	Created restart resilience, liquidity awareness, constraint enforcement, and reduced accidental strategy bias.
Refinements	LLM prompting; fallback parsing; logging and transparency; error handling.	Made the agent more robust against small-model variability and easier to inspect during operation.

Phase 1. Initial Concept & Setup

Item	Cleaned content
User request	"Help building the trading bot."
Parameters established	Alpaca paper trading, stocks only, hourly checks, €10,000 starting capital, 1.5% risk per trade, 10 maximum positions, and 10% daily loss limit.
Deliverables	Configuration management, market data fetching, technical analysis, TinyLlama integration, risk management, order execution, and main orchestration.

Phase 2. Technical Analysis & LLM Integration

Item	Cleaned content
Prompt	Build the technical analysis module and integrate Ollama/TinyLlama for trading decisions.
Deliverables	Minimal technical indicators: RSI, SMA, Bollinger Bands, trend, and volume. Trading-specific prompt engineering. Fallback parsing when the model does not return valid JSON.

```
Prompt format: You are a day trading analyst. Analyze this market data and provide a trading decision. Respond only with JSON containing decision, confidence, and reasoning.
```

Phase 3. Risk Management & Portfolio Awareness

Item	Cleaned content
Prompt	Implement smart risk management with position sizing, daily loss limits, and stop losses.
Requirements implemented	Risk-based position sizing, 15% maximum position value, 3% stop loss, daily loss tracking, maximum-position enforcement, and portfolio status reporting.

Phase 4. Confidence-Based Trading Strategy

Item	Cleaned content
Prompt	Use confidence ratings to decide what to buy, sell, or hold, with the overall goal of growing the portfolio. Consult the LLM when a profitable sell decision requires judgment.
Decision logic	BUY at confidence ≥ 0.65. SELL at confidence ≥ 0.70. Loss-making positions can be closed immediately. Profitable positions trigger a separate LLM profit-taking check. HOLD below the confidence threshold.
Impact	Moved the agent from raw recommendations to controlled trade execution with explicit confidence gates.

Phase 5. Market Hours Validation

Item	Cleaned content
Prompt	Add a market-open check so no trading occurs outside valid trading hours.

Implementation	Check US Eastern Time, allow trading only between 9:30 AM and 4:00 PM, exclude weekends, skip trading cycles when closed, and log when trading will resume.
Impact	Prevents invalid orders outside market hours and makes runtime behavior easier to understand.

Phase 6. Portfolio Synchronization

Item	Cleaned content
Prompt	On restart, check the current Alpaca portfolio before trading resumes.
Implementation	Fetch existing positions from Alpaca, load them into the risk manager, calculate entry prices from cost basis, set stop-loss levels, display P&L, and resume trading with full portfolio awareness.
Impact	Makes the agent restart-safe and prevents it from behaving as if it holds no positions after a reboot.

Phase 7. Dynamic Portfolio Rebalancing

Item	Cleaned content
Prompt	When the agent reaches 10 positions, allow it to sell and buy other stocks instead of becoming static.
Implementation	When the maximum position count is reached and a high-confidence BUY appears, identify the worst-performing position, close it if it is losing money, and open the new high-conviction position.
Impact	Prevents portfolio stagnation and enables intelligent position cycling while respecting maximum-position limits.

Phase 8. Budget Increase & Risk Scaling

Item	Cleaned content
Prompt	“Set the budget to 60000.”
Changes	Starting balance increased from €10,000 to €60,000; risk per trade set to €900; daily loss limit set to €6,000; position sizing scales proportionally.
Impact	Scaled the paper-trading setup while preserving the same percentage-based risk model.

Phase 9. Buying Power Awareness

Item	Cleaned content
Prompt	“Are the get_buying_power and get_account_info methods present in the current build?”
Implementation	Added buying-power and account-info checks covering cash, buying power, portfolio value, and equity.
Impact	Prevents overleveraging and warns when buying power drops below safe operating levels.

Phase 10. Randomized Analysis Order

Item	Cleaned content
Prompt	“Start analyzing a share at random for each cycle, and make sure there are no accidental patterns in the trading strategy.”
Implementation	Copy the configured trading symbols, shuffle the order at the start of every cycle, and analyze each symbol in the randomized order.
Impact	Eliminates sequence bias, such as always analyzing AAPL first.

```
symbols_to_analyze = config.TRADING_SYMBOLS.copy()
random.shuffle(symbols_to_analyze)
for symbol in symbols_to_analyze:
    analyze_symbol(symbol)
```

Phase 11. Extended News Analysis & Pre-Market Strategy

Item	Cleaned content
Prompt	Build a feature where the agent reads more news between market close and market open, creates an opening trade strategy, and then continues with the regular hourly cycle.
Implementation	Fetch 15 articles per symbol, analyze all symbols overnight, persist the strategy, execute once at 9:30 AM ET, and then transition to hourly trading.
Features	Market bias detection, confidence averaging, once-per-day execution, duplicate-prevention flag, and seamless handover to the regular cycle.
Impact	Gives the agent a broader overnight information base before the market opens.

```
You are a pre-market analyst reviewing overnight news for [symbol].

[Extended news summary with 15+ articles]

Based on all this overnight news, what should our opening strategy be?

Consider:
- Major themes in the news
- Sentiment direction: bullish, bearish, or neutral
- Expected volatility
- Opening bias: BUY, SELL, HOLD, or WATCH

Respond with only:
Decision: BUY or SELL or HOLD or WATCH
Confidence: 0.0 to 1.0
Strength: Strong, Moderate, or Weak
Reasoning: one sentence summary
```

Phase 12. Final Capital Configuration

Parameter	Final value
Starting balance	€60,000
Risk per trade	€900, or 1.5%
Daily loss limit	€6,000, or 10%
Maximum positions	10

Appendix C. Testing Guide

Overview

The test suite validates the major components of the news-based trading agent, including technical analysis, decision parsing, risk management, news fetching, LLM response handling, and end-to-end workflows.

Test area	Purpose
Technical indicators	Validate RSI, trend, volume, and indicator formatting.
Decision parsing	Convert natural-language model output into BUY, SELL, or HOLD decisions.
Risk management	Verify position sizing, stop losses, P&L, and operating limits.
News fetching	Retrieve, limit, and format articles for LLM analysis.
Ollama agent	Handle JSON responses, fallback text parsing, and timeout scenarios.
Integration tests	Validate end-to-end position workflows and confidence threshold behavior.

Running Tests

Environment	Command

Mac — all tests	`cd ~/trading-agent`
	`source venv/bin/activate`
	`python -m unittest discover -s . -p "test_*.py" -v`

Mac — specific test	`python test_news_based_agent.py TestMinimalTechnicals -v`

Raspberry Pi — test script	`ssh llmpi.local`
	`cd ~/trading-agent`
	`source venv/bin/activate`
	`bash run_tests.sh`

Raspberry Pi — direct run	`ssh llmpi.local`
	`cd ~/trading-agent`
	`source venv/bin/activate`
	`python test_news_based_agent.py`

Test Coverage

Test class	Coverage
TestMinimalTechnicals	Valid data returns indicators; insufficient data returns none; null input handled gracefully; LLM formatting works.
TestParseDecisionFromText	Parses explicit BUY, SELL, HOLD; infers sentiment-based decisions; extracts confidence values; bounds confidence to 0–1.
TestRiskManager	Opens positions under limits; prevents over-limit trading; enforces daily loss limits; calculates position size, stop loss, P&L, and portfolio status.
TestNewsFetcher	Fetches yfinance news, respects article limits, and formats news for LLM analysis.
TestOllamaAgent	Handles JSON responses, fallback parsing, and timeout behavior.

TestIntegration	Validates end-to-end position lifecycle and confidence-threshold filtering.

Expected Test Output

```
Running News-Based Stock Trading Agent Tests
========================================

test_calculate_position_size (test_news_based_agent.TestRiskManager) ... ok
test_calculate_stop_loss (test_news_based_agent.TestRiskManager) ... ok
test_parse_buy_signal (test_news_based_agent.TestParseDecisionFromText) ...
ok
test_parse_percentage_confidence
(test_news_based_agent.TestParseDecisionFromText) ... ok
...

Ran 25 tests in 0.234s

OK
```

Manual Testing Checklist

Check	Command or verification
News fetching works	`python -c "from news_fetcher import NewsFetcher; nf = NewsFetcher(); print(nf.get_news_for_symbol('AAPL', 1))"`
Technical indicators work	`python test_agent.py`
Ollama responds	`curl http://llmpi.local:11434/api/tags`
Alpaca connection works	Run the Alpaca test in **test_agent.py**.
Agent analyzes a symbol	Run the agent once manually and verify the first output.
Risk limits are enforced	Verify position size does not exceed account limits.

Key Test Scenarios

Scenario	Input conditions	Expected result
Strong BUY signal	Three bullish articles, RSI below 30, uptrend.	BUY with 80%+ confidence.
Strong SELL signal	Three bearish articles, RSI above 70, downtrend.	SELL with 80%+ confidence.
Mixed signal	One bullish article, two neutral articles, RSI around 50, neutral trend.	HOLD with 50–60% confidence.
Risk management	€10,000 starting balance, €150 risk per trade, 10 maximum positions, €1,000 daily loss limit.	Stop trading after 10 positions or €1,000 loss.

Debugging Failed Tests

Failure area	Likely check
MinimalTechnicals	Confirm pandas-ta is installed and ensure at least 14 data points are available for RSI calculation.
OllamaAgent	These tests use mocks and do not require a real Ollama connection; check unittest.mock imports if mocking fails.
RiskManager	Check config.py values and reset STARTING_BALANCE to the expected test default where needed.
NewsFetcher	Live fetching requires internet access; mocked tests do not require internet.

Performance Targets and Continuous Testing

Topic	Guidance
Expected runtime	Tests should complete in under 2 seconds. If they exceed 5 seconds, check CPU load, network saturation, and test data size.

Hourly scheduled testing	`ssh llmpi.local`
	`crontab -e`
	`0 * * * * cd ~/trading-agent && source venv/bin/activate && python test_news_based_agent.py >> tests.log 2>&1`

Bug report contents	Include the failing test name, full traceback, environment, and recent changes.

```
TEST: TestOllamaAgent.test_get_trading_decision_json_response
ERROR: AssertionError: 'BUY' != 'HOLD'
ENV: Pi 3, Python 3.13, ollama_agent.py modified
```

Appendix D. Complete Testing Overview

Testing Strategy Summary

Metric	Value
Total test suites	3
Total tests	76+
Coverage	Core agent features, new features, and pre-market strategy.
Pass rate	100%
Test types	Unit tests, integration tests, and logic verification.

Test Suites

Suite	Tests	Status	Coverage focus
Core Agent Features	29	All passing	Technical indicators, decision parsing, risk management, news fetching, Ollama handling, and end-to-end lifecycle.
New Features	24	All passing	Market hours, risk manager behavior, confidence thresholds, technical indicators, portfolio status, randomization, account info, and LLM confidence handling.
Pre-Market Feature	23	All passing	Extended news fetching, strategy structure, pre-market timing, strategy execution, bias calculation, persistence, and news-volume handling.

Coverage by Component

Component	Suite	Coverage	Status
Technical analysis	Core agent	4 tests	100%
Decision parsing	Core agent	9 tests	100%
Risk management	Core and new features	16 tests	100%
News fetching	Core and pre-market	6 tests	100%
LLM integration	Core agent	3 tests	100%
Market hours	New features	3 tests	100%
Confidence logic	Core and new features	5 tests	100%
Pre-market analysis	Pre-market feature	23 tests	100%

Running Tests

Task	Command

Run all suites	`cd ~/trading-agent`
	`source venv/bin/activate`
	`python -m unittest discover -s . -p "test_*.py" -v`

Run core agent tests	`python -m unittest test_news_based_agent -v`
Run new feature tests	`python -m unittest test_new_features_simple -v`
Run pre-market tests	`python -m unittest test_pre_market_feature -v`
Run with coverage	`pip install coverage` `coverage run -m unittest discover`

	`coverage report` `coverage html`

Test Results Summary

Suite	Tests	Execution time	Pass rate
test_new_features_simple.py	24	0.161 seconds	100%
test_pre_market_feature.py	23	0.610 seconds	100%
test_news_based_agent.py	29	~1.5 seconds	100%
Total suite	76	~2.3 seconds	100%

Mocking Strategy and Edge Cases

Area	Testing approach
Mocked components	Alpaca API, yfinance, Ollama/TinyLlama, and current time.
Why mock?	Isolation, speed, repeatability, safety, and easy simulation of edge cases.
Edge cases	Low data, LLM timeout, parse failure, market closed, maximum positions, daily loss limit, low confidence, insufficient capital, empty news, and duplicate strategy execution.
Production-only verification	Actual Alpaca API calls, real news fetching, full LLM inference on Pi3, multi-day runs, live market stress conditions, and concurrent position management.

Continuous Testing and Maintenance

Topic	Guidance
Pre-commit hooks	`git pre-commit install`

CI/CD pipeline	`test:`
	`  script:`
	`  - source venv/bin/activate`
	`  - python -m unittest discover -v`
	`  coverage: '/^TOTAL\s+.*?\s+(\d+%)$/'`

Adding tests	Create the test in the relevant suite, use descriptive names, keep tests isolated, and run the full suite after adding.
Updating mocks	Update the relevant mock, run the affected suite, and add edge-case tests when API behavior changes.
Performance monitoring	`python -m pytest -v --durations=10` `python -m cProfile -m unittest test_new_features_simple`

Appendix E. Test Results

Overview

Comprehensive test results for the new trading-agent features added since the previous test suite.

Area	Tests	Status	Key validation
Market hours validation	3	Passing	9:30 AM open, 4:00 PM close, weekday/weekend detection.
Risk manager functionality	7	Passing	Position storage, P&L, max positions, daily loss limit, stop loss, sizing, and portfolio status.
Confidence-based trading	3	Passing	BUY ≥ 0.65, SELL ≥ 0.70, HOLD below threshold.
Technical indicators	3	Passing	Minimum data requirement, required fields, and RSI bounds.
Portfolio management	2	Passing	Status structure and position count.
Randomization logic	2	Passing	Symbol preservation and varied analysis order.
Account info structure	3	Passing	Cash, buying power, portfolio value, equity, and liquidity validation.
LLM confidence handling	2	Passing	Confidence bounds and correlation with strong signals.

Features Tested

Feature	Validation result
Market open check	Validates US market hours, blocks weekends, and blocks before/after market trading.
Portfolio sync on startup	Loads existing positions, calculates entry prices from cost basis, and estimates stop losses.
Dynamic rebalancing	Identifies worst-performing positions, closes losers for high-confidence opportunities, and respects the maximum position limit.
Confidence-based trading	Applies BUY and SELL thresholds and holds on low-confidence signals.
Buying-power checks	Implements get_buying_power() and get_account_info(), preventing trades when liquidity is constrained.
Randomized analysis	Shuffles symbol order each cycle, removes sequence bias, and ensures no symbols are skipped.
Stop-loss enforcement	Closes positions 3% below entry and protects against runaway losses.

Test Execution

```
Ran 24 tests in 0.161s

OK
```

All tests passed. No failures or errors were reported.

Coverage by Component

Component	Coverage	Status
Market hours	100%	Passing
Risk management	100%	Passing
Position sizing	100%	Passing
Stop loss	100%	Passing
Confidence thresholds	100%	Passing
Technical indicators	100%	Passing
Portfolio sync	Unit tested	Passing
Buying power	Unit tested	Passing
Randomization	100%	Passing
LLM integration	Logic tested	Passing

Integration Status

Integration check	Result
Market check	Blocks trading outside hours.
Portfolio sync	Loads real positions.
Confidence thresholds	Gate all trades.
Risk limits	Prevent over-trading.
Buying power	Prevents under-capitalized positions.
Stop losses	Protect downside risk.

Running Tests

Task	Command

Run the local test suite	`cd ~/trading-agent`
	`source venv/bin/activate`
	`python -m unittest test_new_features_simple -v`

Run with coverage	`pip install coverage`
	`coverage run -m unittest test_new_features_simple`
	`coverage report`

Next Steps and Test Files

Category	Details
Next steps	Integration tests with mocked Alpaca API; end-to-end trading scenario tests; performance benchmarks; stress tests with edge cases.
Test files	test_new_features_simple.py: simplified unit tests, 24 passing. test_new_features.py: full feature tests with API mocking. test_news_based_agent.py: 29 core agent tests.
Test run date	2026-07-16
Environment	macOS, Python 3.13, trading-agent virtual environment.
Overall result	All tests passing.